Great Principles of Computing

The great principles of computing have been interred beneath layers of technology in our understanding and our teaching. It is time to set them free.

Computer science was born in the mid-1940s with the construction of the first electronic computers. In just 60 years, computing has come to occupy a central place in science, engineering, business, and everyday life. Many whose lives are touched by computing want to know how computers work and how dangerous or risky they are; some want to make a profession from working with computers; and most everyone asks for an uncomplicated framework for understanding this complex field. Can their questions be answered in a compact, compelling, and coherent way?

In what follows, I will answer affirmatively, offering a picture of the great principles of computing. There are two kinds: principles of computation structure and behavior, which I call mechanics, and principles of design. What we call principles are almost always distilled from recurrent patterns observed in practice. Do practices shape to underlying principles? Do principles shape to practice? It is impossible to tell. In

my description, therefore, I portray principles and practices as two equal dimensions of computing.

A principles-based approach is not new to science. The mature disciplines such as physics, biology, and astronomy portray themselves with such an approach. Each builds rich structures from a small set of great principles. Examples of this approach are

Lectures in Physics by Richard Feynman [4], *The Joy of Science* by Robert Hazen and James Trefil [5], and *Cosmos* by Carl Sagan [7]. Newcomers find a principles-based approach to be much more rewarding because it promotes understanding from the beginning and shows how the science transcends particular technologies.

In my portrait, the contexts of use and their histories are imbued into principles, computing practices, and core technologies. Indeed, you cannot understand a principle without knowing where it came from, why it is important, why it is recurrent, why it is universal, and why it is unavoidable. Numerous application domains have influenced the design of all our core technologies. For example, the different styles of the languages Ada, Algol, Cobol, C++, Fortran, HTML, Java, Lisp, Perl, Prolog, and SQL flow out of the application domains that inspired them. You cannot make sense of the debates about the limits of machine intelligence without understanding cognitive science and linguistic philosophy. In software, unless you understand the

The Profession of IT

The principles of a field are actually a set of interwoven stories about the structure and behavior of field elements.

different ways engineers and architects use the term "design," you cannot make sense of the tug-of-war between traditionalists promoting systems produced by a highly methodical engineering process, and agile developers promoting systems built for customer satisfaction, artistry, good taste, simplicity, and elegance.

The following sections describe principles of mechanics, principles of design, and computing practices as a framework supporting core technologies and application domains. A few implications of the framework on organization and content of computing curricula, and on the profession itself, are discussed at the end of this column.

Mechanics

In the 1950s, our field's founders portrayed their young science as a set of core technologies that supported application domains. They listed their core technologies as algorithms, numerical methods, computation models, compilers, languages, and logic circuits. Over the next 30 years, we added a few more: operating systems, information retrieval,

algorithms	management information systems
artificial intelligence	natural-language processing
compilers	networks
computational science	operating systems
computer architecture	parallel computation
data mining	programming languages
data security	real-time systems
data structures	robots
databases	scientific computation
decision support systems	software engineering
distributed computation	supercomputers
e-commerce	virtual reality
graphics	vision
human-computer interaction	visualization
information retrieval	workflow

Table 1. Core technologies of computing.

databases, networks, artificial intelligence, human-computer interaction, and software engineering. The 1989 ACM/IEEE report, *Computing as a Discipline*, listed nine core technology areas [2]. Since then, the total number of core technology areas has tripled (see Table 1). Today, learning the mechanics of these technologies and their hundreds of possible direct interactions has become a daunting challenge.

In an effort to stem "curriculum bloat" from this growth, the *Curriculum 2001* report emphasizes the ideas at the intersection of the core technologies [3]. Unfortunately, a list of core technologies

and their great ideas does little to convey the great principles of computing. Two books seeking to popularize computing focus on a few "great ideas" in a few of these areas, but their coverage is far from complete [1, 6]. Neither of these authors discusses which ideas are fundamental principles of all the core technologies.

Locating the fundamental principles of the field looks, therefore, to be a very attractive project. It calls to mind a picture in which the principles are the foundation of a pantheon with one pillar for each great principle. Unfortunately, as we shall soon see, such a picture is an unsatisfactory portrayal of computing.

Our initial question is: How shall we express our principles? It seems like we are looking for declarative statements, such as:

"The Turing machine is a model of universal computation."

"All information can be encoded as strings of bits."

"The number of bits in a message source is given by its entropy."

But this approach quickly becomes contentious. Some people argue over the definitions of terms like computation, informa-

tion, or message sources. Others ask whether some of the words ought to be qualified—such as algorithmic computation, physically represented bits, or discrete message sources. Still others ask why these statements are singled out and not others, such as "Every function imposes a lower-bound running time on all algorithms that compute it." Most everyone demands statements of obvious relevance to the familiar core technologies. But they wrestle over the selection criteria for principle statements, such as universality, recurrence, invariance, utility for prediction, or scope of consequences.

How do other fields express their principles? Physicists use terms like photons, electrons, quarks, quantum wave function, relativity, and energy conservation. Astronomers use terms like planets, stars, galaxies, Hubble shift, and black holes. Thermodynamicists uses terms like entropy, first law, second law, and Carnot cycle. Biologists use terms like phylogeny, ontogeny, DNA, and enzymes. Each of these terms is actually *the title of a story!* The principles of a field are actually a set of interwoven stories about the structure and behavior of field elements. They are the names of chapters in books about the field [4, 5, 7].

These principle-stories seek to make simple the complex history of a complex area. They tell history, showing how the principle evolved and grew in acceptance over time. They name the main

contributors. They chronicle feats of heroes and failures of knaves. They lay out obstructions and how they were overcome. They

Window	Central Concern	Principal Stories
Computation	What can be computed; limits of computing.	Algorithm, control structures, data structures, automata, languages, Turing machines, universal computers, Turing complexity, Chaitin complexity, self-reference, predicate logic, approximations, heuristics, non-computability, translations, physical realizations.
Communication	Sending messages from one point to another.	Data transmission, Shannon entropy, encoding to medium, channel capacity, noise suppression, file compression, cryptography, reconfigurable packet networks, end-to-end error checking.
Coordination	Multiple entities cooperating toward a single result.	Human-to-human (action loops, workflows as supported by communicating computers), human-computer (interface, input, output, response time); computer-computer (synchronizations, races, deadlock, serializability, atomic actions).
Automation	Performing cognitive tasks by computer.	Simulation of cognitive tasks, philosophical distinctions about automation, expertise and expert systems, enhancement of intelligence, Turing tests, machine learning and recognition, bionics.
Recollection	Storing and retrieving information.	Hierarchies of storage, locality of reference, caching, address space and mapping, naming, sharing, thrashing, searching, retrieval by name, retrieval by content.

Table 2. The five windows of computing mechanics.

explain how the principle works and how it affects everything else. The game is to define many terms in terms of a few terms and to logically derive many statements from a few statements.

Astronomy, thermodynamics, and physics use the term *mechanics* for the part of their fields dealing with the behavior and structure of components. For example, Celestial Mechanics deals with the motions of heavenly bodies; Statistical Mechanics with the macro behavior of physical systems comprising large numbers of small particles; Quantum Mechanics with

wave behaviors of subatomic particles; Rigid-Body Mechanics with the balance of forces within and between connected objects. I

adopt this term for computing.
Computing Mechanics deals with the structure and operation of computations. It does so with stories for algorithm, Turing machine, grammar, message entropy, process, protocol stack, naming, caching, machine learning, virtual machine, and more. I found I could group the stories into the five categories of computation, communication, coordination, automation, and recollection (see Table 2). Every core technology expresses all five in its own way.

The lines between these categories are blurry. For example, the Internet protocol stack is an element of both communication and

coordination; naming and caching are both elements of communication and recollection. Therefore, I found it better to view the categories as windows into computing mechanics (see Figure 1). Although the views through the edges of windows overlap, the view through the centers is distinctive.

Design

Computing Mechanics does not exhaust all the principles of our field. Computing professionals follow principles of design that enable them to harness mechanics in the service of users and customers. Five concerns drive the design principles:

• Simplicity: Various forms of abstraction and structure that overcome the apparent complexity of the applications.

• Performance: predicting throughput, response time, bottlenecks, capacity planning.

• Reliability: redundancy, recovery, checkpoint, integrity, system trust.

• Evolvability: adapting to changes in function and scale.

• Security: access control, secrecy, privacy, authentication, integrity, safety.

The design principles themselves include abstraction, information hiding, modules, separate compilation, packages, version control, divide-and-conquer, functional levels, layering, hierarchy, separation of concerns, reuse,

encapsulation, interfaces, and virtual machines. These principles are *conventions* that we collectively have found to lead consistently to dependable and useful programs, systems, and applications. These conventions are practiced within constraints of cost, schedule, compatibility, and usability.
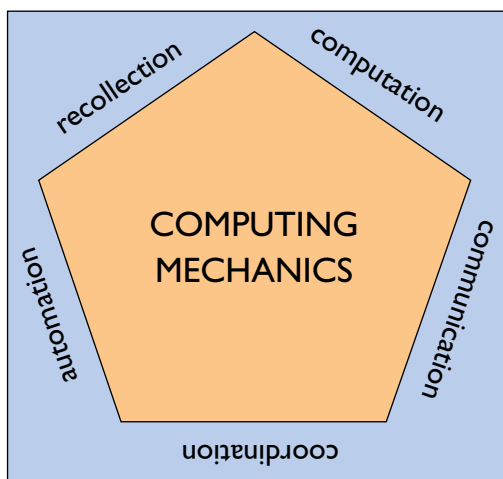

Figure 1. The five windows.

Design is not the same in computing as it is in other fields. In computing we design *abstract objects* that *perform actions*. Other fields use abstraction to explain or to organize tangible objects. Since design tells us about arrangements of basic components, design sits above mechanics in our picture of the field.

Might we call Computing Mechanics the "science" of computing and the Design Principles the "art"? I think not. There is good science and engineering and much art at all levels—mechanics, design, and applications.

Computing Practices

Our picture of computing needs more than mechanics and design. It needs an account of the computing practices that characterize our skills as professionals. Our competence is judged not by our ability to explain principles, but by the quality of what we do. I found five main categories of computing practice:

• Programming: Using programming languages to build software systems that meet specifications created in cooperation with the users of those systems. Computing professionals must be multilingual, facile with the numerous programming languages, each attuned to its own strategies for solving problems.

• Engineering Systems: Designing and constructing systems of software and hardware components running on servers connected by networks. These practices include a design component concerned with organizing a system to produce valuable and tangible benefits for the users; an engineering component concerned with the modules, abstractions, revisions, design decisions, and risks in the system; and an operations component concerned with configuration, management, and maintenance of the system. High levels of skill are needed for large programmed systems encompassing thousands of modules and millions of lines of code.

• Modeling and validation: Building models of systems to make predictions about their behavior under various conditions;

Our competence is judged not by our principles, but by the quality of what we do.

and designing experiments to validate algorithms and systems.

• Innovating: Exercising leadership to design and bring about lasting changes to the ways groups and communities operate. Innovators watch for and analyze opportunities, listen to customers, formulate offers customers see as valuable, and manage commitments to deliver the promised results. Innovators are history-makers who have strong historical sensibilities.

• Applying: Working with practitioners in application domains to produce computing systems that support their work. Working with other computing professionals to produce core technologies that support many applications.

I cannot overemphasize the importance of including computing practices in a portrait of our field. If we adopt a picture that ignores practices, our field will end up like the failed "new math" of the 1960s—all concepts, no practice, lifeless; dead.

Our portrait is now complete (see Figure 2). It consists of computing mechanics (the laws and universal recurrences that govern the operation of computations), design principles (the conventions for designing computations), computing practices (the standard ways of building and deploying

computing systems), and core technologies (organized around shared attributes of application domains). Although not shown in the figure, the entire framework floats in a rich contextual sea of application domains, collectively

Levels	Central Questions	Example Technologies
Application Domains	How do we work with others to design computing that serves them?	Supercomputers, grid computing, domain databases, graphics design, interfaces, …
Core Technologies	How do we design computations that support common elements across applications?	Algorithms, databases, networks, operating systems, HCI, AI, …
Design Principles	How do we organize ourselves to build computations that work?	Design tools, object-oriented programming, layering, virtual machines, authentication, …
Computing Mechanics	How do computations work?	Logic simulators, protocol stack, workflow, expert systems, virtual memory, …

Table 3. Levels of action in computing practices.

exerting strong influences on core technologies, design, mechanics, and practice. Each level of the picture has a characteristic question that justifies its place in the hierarchy and exposes the integral role of practices (see Table 3).

Implications

By aligning with traditions of other science fields, a portrait of computing organized around great principles and practices promotes greater understanding of the science and engineering behind information technology. It significantly improves our abil-

ity to discuss risks, benefits, capabilities, and limitations with people outside the field. It recognizes that computing is action-oriented and has many customers, and that the context in which computing is used is as important as

the mechanics of computing. It also clarifies professional competence, which depends on dexterity with mechanics, design, practices, core technologies, and applications.

For years, many others have seen our field as programming. Through our 1989 *Computing as a Discipline* report [3] we hoped to encourage new curricula that would overcome this misleading image. But this was not to be. Our practice of embedding a programming language in the first courses, started when languages were easy for beginners, has created a monster. Our students are being overwhelmed by the complexities of languages that many experts find
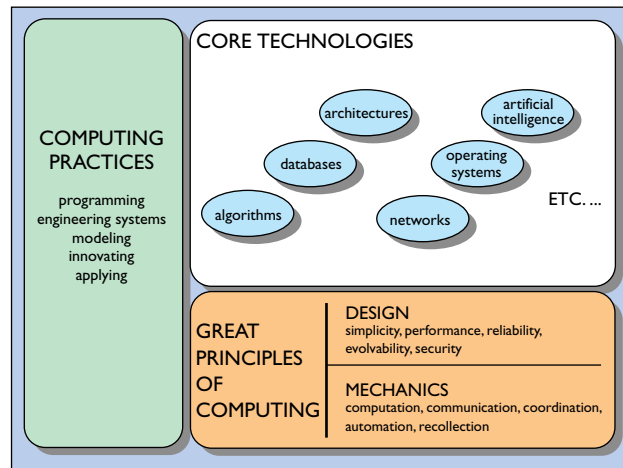
The Profession of IT

**Figure 2.
Principles-based
portrait of
computing.**

CORE TECHNOLOGIES

architectures

artificial
intelligence

databases

operating
systems

algorithms

networks

ETC. ...

COMPUTING
PRACTICES

programming
engineering systems
modeling
innovating
applying

GREAT
PRINCIPLES
OF
COMPUTING

DESIGN
simplicity, performance, reliability,
evolvability, security

MECHANICS
computation, communication, coordination,
automation, recollection

challenging (typically Java and C++). Many students have turned to cheating and plagiarism as ways to pass these courses, and 35%–50% drop out prematurely. Many do not experience the joy of computing: the interplay between the great principles, the ways of algorithmic thinking, and the solutions of interesting problems. A curriculum organized around the framework offered here may rescue us from this unfortunate predicament. The current first courses (CS1, CS2, ...) can be replaced by Computing Mechanics (CM1, CM2, ...) and their extensive programming content can be moved to Programming Practices courses (PP1, PP2, ...) embedded within a larger Computing Practices track. The standard core courses (for example, algorithms, operating systems, databases, software engineering, or networks) can then be reshaped to extend computing mechanics into their areas rather than teaching applicable mechanics from scratch. (Aside to academic colleagues: starting with computing mechanics is not a "breadth-first"

approach; the framework promotes depth in concepts, design, and practice.)

It is time for us to make ourselves known by saying our mechanics, our design principles, and our practices. It is time to stop hiding the enormous depth and breadth of our field. **C**

https

REFERENCES
1. Biermann, A. *Great Ideas in Computer Science, 2nd ed.* MIT Press, 1997.
2. Denning, P., et al. Computing as a discipline. *Commun. ACM 32*, 1 (Jan. 1989), 9–23.
3. *Curriculum 2001 Final Report*; computer.org/education/cc2001/final/.
4. Feynman, R. *Lectures in Physics*. Addison-Wesley, 1970.
5. Hazen, R. and Trefil, J. *Science Matters*. Anchor, 1991.
6. Hillis, D. *The Pattern on the Stone*. Basic Books, 1999.
7. Sagan, C. *Cosmos*. Random House, 2002.

PETER DENNING (pjd@nps.navy.mil) is past president of ACM.